Death by a thousand cuts

(worst-case execution time by bounded model checking)

David Monniaux

CNRS / VERIMAG

June 17, 2013

Joint work with Julien Henry, Claire Maïza and Diego Caminha

A typical control system

```
initialize();
while(true) {
  loop_body();
  wait_for_next_clock_tick();
}
```

The **WCET** (worst-case execution time) of loop_body() must be less than the period of the clock (- safety margin).

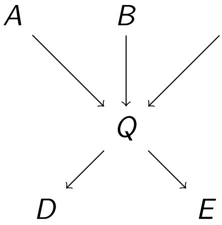
Usual approach

1. Run an abstract interpretation / static analysis for e.g. pointer analysis, indirect control flow, value ranges.
2. Use results of 1. to perform cache / pipeline analysis over model of architecture.
3. Derive WCET for each basic block of the program.
4. Reassemble WCET for whole program by integer linear programming (ILP) using maximal iteration counts (1.) and block WCET (3.).

In this talk: discuss 4. and improvements.

Setting up integer linear programming

Along an execution: $x_{\alpha\beta}$ count of times it goes through control edge $\alpha \to \beta$



"Kirchhoff's circuit law": $x_{aq} + x_{bq} + x_{cq} = x_{qd} + x_{qe}$

Combine with inequalities from value range analysis on loop counters, e.g. $x_{aq} \leq 10$.

Solving ILP

Total time is bounded by $T = \sum_{\alpha,\beta} T_{\alpha\beta} x_{\alpha\beta}$ where $T_{\alpha\beta}$ is "local WCET" for block $\alpha \to \beta$.

Maximize T subject to the Kirchhoff and bound constraints.

Example: OTAWA tool from IRIT

Note: for the above simple constraints, ILP=LP.

Possibility of adding more refined constraints, e.g. $2x_{AB} + x_{BC} = 100$.

Loop-free programs

In typical control applications:

- one main big control loop
- smaller internal loops, with syntactically constant bounds (e.g. `for ( int i=0; i<100; i++) { ... }`)

Solution: **unroll** internal loops and get a loop-free program for WCET

(NB: the Astrée static analyzer roughly does the same for proving safety)

The ILP approach on loop-free programs

(Without additional constraints:)
amounts to finding the **longest path** in a DAG from initial to final control state.

No need for LP/ILP, a simple linear-time graph traversal is sufficient.

How about **semantic constraints**?

Semantic constraints

A control application typically has some invariants such as "modes A and B are exclusive"
(e.g. in avionics "take-off mode and landing mode are exclusive")

But application code may look like:

```
if (take_off_mode) {
  /* A */
}
...
if (landing_mode) {
  /* B */
}
```

Syntactic WCET will count $T_A + T_B + T_{rest}$
taking into account the semantics: $\max(T_A, T_B) + T_{rest}$.

Bounded model-checking for WCET

Take program P, precondition F_{pre}, postcondition F_{post}

SMT-solving:

Solve $F_{pre} \wedge \llbracket P \rrbracket \wedge F_{post}$, solution is an execution trace τ

τ has Booleans $x_{ab} \in \{0, 1\}$, maximize $T^* = \sum T_{ab} x_{ab}$

Optimization modulo SMT

Binary search

Maintain an interval $[l, h]$ containing T^* (initialize $l = 0$ and $h =$ some upper bound)

- test whether there exists a trace of total time $B \geq \frac{l+h}{2}$
- halve $[l, h]$ accordingly and restart until $l = h$

Sounds simple, no?

A really simple example

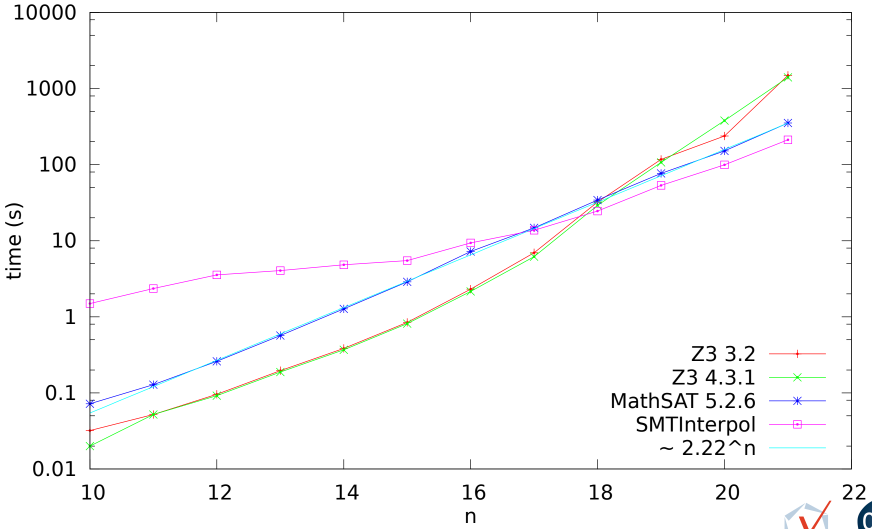
$b_1, \ldots, b_n$ unconstrainted nondeterministic choices

```
if (b1) { /* timing 2 */ } else { /* timing 3*/ }
if (b1) { /* timing 3 */ } else { /* timing 2*/ }
...
if (bn) { /* timing 2 */ } else { /* timing 3*/ }
if (bn) { /* timing 3 */ } else { /* timing 2*/ }
```

"Obviously" all traces take time $5n$.

Proving optimality is costly

Proving that there is no trace longer than $B = 5n$



Cost **exponential** in n.

Why such high cost for diamonds?

(Recall: the property to prove is "trivial" by human inspection.)

Formula $x_1 = ite(b_1, 2, 3) \wedge y_1 = ite(b_1, 3, 2) \wedge \cdots \wedge x_n = ite(b_n, 2, 3) \wedge y_n = ite(b_n, 3, 2) \wedge x_1 + y_1 + \cdots + x_n + y_n \geq 5n$

A SMT-solver based on "DPLL(T)" enumerates a Boolean choice tree over $b_1, \ldots, b_n$, cutting branches when encountering **inconsistent numerical constraints**.

What are the possibly inconsistent numerical constraints here?

Why such high cost for diamonds?

(Recall: the property to prove is "trivial" by human inspection.)

Formula $x_1 = ite(b_1, 2, 3) \wedge y_1 = ite(b_1, 3, 2) \wedge \cdots \wedge x_n = ite(b_n, 2, 3) \wedge y_n = ite(b_n, 3, 2) \wedge x_1 + y_1 + \cdots + x_n + y_n \geq 5n$

A SMT-solver based on "DPLL(T)" enumerates a Boolean choice tree over $b_1, \ldots, b_n$, cutting branches when encountering **inconsistent numerical constraints**.

What are the possibly inconsistent numerical constraints here?
All of the form
$x_1 \leq ? \wedge y_1 \leq ? \wedge \ldots x_n \leq ? \wedge y_n \leq ? \wedge x_1 + y_1 + \cdots + x_n + y_n \geq 5n$.

Thus of size $2n + 1$.

Why such high cost for diamonds?

(Recall: the property to prove is "trivial" by human inspection.)

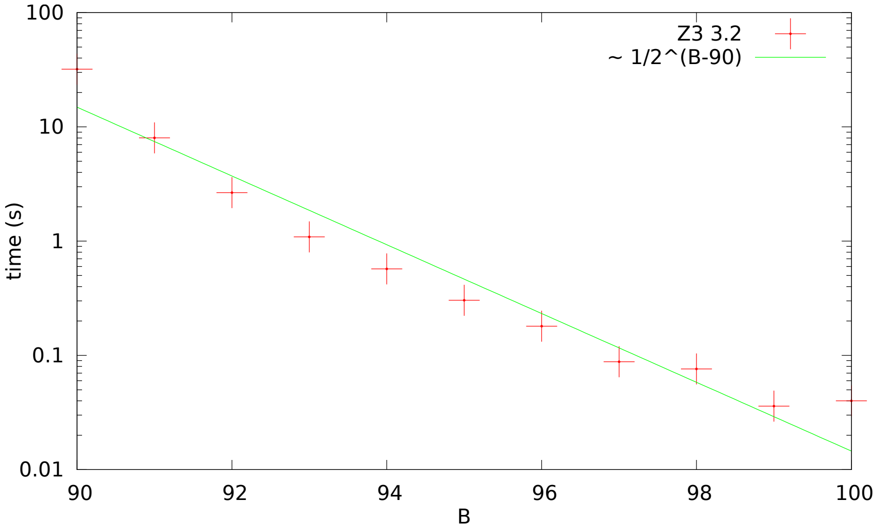
Formula $x_1 = ite(b_1, 2, 3) \wedge y_1 = ite(b_1, 3, 2) \wedge \cdots \wedge x_n = ite(b_n, 2, 3) \wedge y_n = ite(b_n, 3, 2) \wedge x_1 + y_1 + \cdots + x_n + y_n \geq 5n$

A SMT-solver based on "DPLL(T)" enumerates a Boolean choice tree over $b_1, \ldots, b_n$, cutting branches when encountering **inconsistent numerical constraints**.

What are the possibly inconsistent numerical constraints here?
All of the form
$x_1 \leq ? \wedge y_1 \leq ? \wedge \ldots x_n \leq ? \wedge y_n \leq ? \wedge x_1 + y_1 + \cdots + x_n + y_n \geq 5n$.

Thus of size $2n + 1$.
2^n of them. The solver has to prove them inconsistent one by one.

Binary search with such high costs



Tolerable away from the optimum, but cost grows exponential close to the optimum.

Solutions?

Diamond formulas are a known issue with DPLL(T); solutions proposed by Cotton and McMillan, but implemented in no mainstream tool.

Solutions?

Diamond formulas are a known issue with DPLL(T); solutions proposed by Cotton and McMillan, but implemented in no mainstream tool.

Instead of solving it in the SMT-solver, fix it **in the encoding**.

A remark

Human remark: “but **obviously** $x_i + y_i = 5$ for any i”

If these constraints are added to the SMT formula, the problem becomes trivial.

$x_i + y_i \leq 5$ is **implied** by the original formula
“Normal” SMT solvers don't it because **they do not invent predicates**.

Our solution

- Distinguish "blocks" in the program.
- Compute upper bound B_i on WCET for each block i (recursive call or rougher bound)
- Add these bounds to the SMT formula encoding the program ($x_1 + \cdots + x_5 \leq B_1$, $x_6 + \cdots + x_{10} \leq B_2$, etc.)
- Do binary search

Experimentation in progress!

On early examples, adding "cuts" cut SMT time from "nonterminating after one night" to "a few seconds".

Cut

The new constraints

- are implied by the original problem
- but not syntactically present in it
- speed up the computation

In operation research, such constraints are referred to as cuts.

Perspectives and Variants

Counterexample refinement loop for ILP (Pascal Raymond)

Generalization to static analysis outside of WCET (Julien Henry + DM)

Gratuitous announcements

Static analysis tool (generates SMT formulas and invariants out of LLVM): `http://pagai.forge.imag.fr`

ERC project `http://stator.imag.fr` (**postdoc positions**)

Polyhedra library `http://verasco.imag.fr/wiki/VPL` (constraint-only, compares favorably to Parma and Apron)